# 并查集及应用

## 一：并查集

是一种数据结构。主要解决元素分组问题，无向图有无环问题。管理一系列不相交的集合，支持2种操作，查找根节点：FindRoot，合并节点：MergeElemet

# 二：模板伪代码

## 最原始的

查：

int FindRoot(int\* parent, int ele)

{

int i;

if (ele != parent[ele]) {

return FindRoot(parent, parent[ele]);

}

return ele;

}

并：

void Merge(int\* parent, int x, int y)

{

int xRoot, yRoot;

xRoot = FindRoot(parent, x);

yRoot = FindRoot(parent, y);

parent[xRoot] = yRoot;

}

## 优化的

/\*路径压缩：查找根结点的时候节点将同一个集合的点的父节点都置为根节点：递归方法实现\*/其实也可以不用记忆，单纯找根节点就可以了

int FindRootCompress(int\* parent, int ele)

{

    if (ele != parent[ele]) {

        parent[ele] = FindRootCompress(parent, parent[ele]);

        return parent[ele];

    }

    return ele;

}

/\*

按秩合并：即将树层次低的往层次高树合，同高度的合并，父节点的秩要加1

\*/

void MergeEleByRank(int\* parent, int\* rank, int x, int y)

{

    int xRoot, yRoot;

    xRoot = FindRootCompress(parent, x);

    yRoot = FindRootCompress(parent, y);

    // 同属于一个集合无需合并

    if (xRoot == yRoot) {

        return ;

    }

    if (rank[xRoot] > rank[yRoot]) {

        parent[yRoot] = xRoot;

    }

    else if (rank[yRoot] > rank[xRoot]){

        parent[xRoot] = yRoot;

    }

    else {

        parent[xRoot] = yRoot;

        rank[yRoot]++;

    }

}

## 3.小实例

int main()

{

    int parent[Vertices] = {0};

    int rank[Vertices] = {0};

int i;

// 6对亲戚关系，先维护并查集

    int relations[][2] = {

        {0,1},

        {1,2},

        {2,5},

        {1,6},

        {3,6},

        {6,4},

    };

    for (i = 0; i < Vertices; i++) {

        parent[i] = i;

        rank[i] = 1;

    }

    for (i = 0; i < sizeof(relations) / sizeof(relations[0]); i++) {

        MergeEleByRank(parent, rank, relations[i][0], relations[i][1]);

}

// 查看6,9是不是亲戚

int 6Root = FindRootCompress(parent, 6);

int 9Root = FindRootCompress(parent, 9);

if (6Root != 9Root) {

printf(“6 and 9 is not relative\n”);

}

    printf("ele %d father is %d\n", 4, res);

    return 0;

}

三：应用及实例